Implementing the AsyncTaskBarrier Framework Using Project Reactor (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand the API of the `AsyncTaskBarrier` class for Project Reactor

---

**Class AsyncTaskBarrier**

```java
public class AsyncTaskBarrier
    extends java.lang.Object

This class asynchronously runs tasks that use the Project Reactor framework and ensures that the calling method doesn't exit until all asynchronous task processing is completed.
```

**Method Summary**

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>register</td>
<td>Register the task task so that it will be run asynchronously when runTasks() is called.</td>
</tr>
<tr>
<td>static void</td>
<td>runTasks()</td>
<td>Run all the register tasks.</td>
</tr>
</tbody>
</table>

---

See Reactive/flux/ex4/src/main/java/utils/AsyncTaskBarrier.java
Learning Objectives in this Part of the Lesson

- Understand the API of the `AsyncTaskBarrier` class for Project Reactor
- Know how to use `AsyncTaskBarrier` in practice

```java
AsyncTaskBarrier.register(this::syncThrowException);
AsyncTaskBarrier.register(this::asyncThrowException);
AsyncTaskBarrier.register(this::syncNoException);
AsyncTaskBarrier.register(this::asyncNoException);

long testCount = AsyncTaskBarrier
    .runTasks()
    .blockingGet();

assertEquals(testCount, 2);
```

See Reactive/flux/ex4/src/test/java/utils/AsyncTaskBarrierTests.java
The AsyncTask Barrier Class API
The AsyncTaskBarrier Class API

- The AsyncTaskBarrier API contains methods that register, unregister, & (a)synchronously run tasks

```java
public class AsyncTaskBarrier {
    private List<Supplier<Mono<Void>>> tasks;

    public AsyncTaskBarrier() {
    }

    public void register(Supplier<Mono<Void>> task) {
    }

    public boolean unregister(Supplier<Mono<Void>> task) {
    }

    public Mono<Long> runTasks() {
        return Mono.empty();
    }
}
```
The AsyncTaskBarrier Class API

- The AsyncTaskBarrier API contains methods that register, unregister, & (a)synchronously run tasks
- It provides methods that register & unregister tasks passed as Suppliers

```java
AsyncTaskBarrier()
register(Supplier<Mono<Void>>): void
unregister(Supplier<Mono<Void>>): boolean
runTasks(): Mono<Long>
```

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html](http://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)
The AsyncTaskBarrier Class API

- The AsyncTaskBarrier API contains methods that register, unregister, & (a)synchronously run tasks
- It provides methods that register & unregister tasks passed as Suppliers
- These tasks are stored in a List

See docs.oracle.com/javase/8/docs/api/java/util/List.html
The AsyncTaskBarrier Class API

- The AsyncTaskBarrier API contains methods that register, unregister, & (a)synchronously run tasks
  - It provides methods that register & unregister tasks passed as Suppliers
  - It also provides a method that runs all registered tasks (a)synchronously
The `AsyncTaskBarrier` API contains methods that register, unregister, & (a)synchronously run tasks.

- It provides methods that register & unregister tasks passed as Suppliers.
- It also provides a method that runs all registered tasks (a)synchronously.
- This method doesn’t block.

```
<<Java Class>>

`AsyncTaskBarrier`

- `sTasks: List<Supplier<Mono<Void>>>`
- `AsyncTaskBarrier()`
- `register(Supplier<Mono<Void>>): void`
- `unregister(Supplier<Mono<Void>>): boolean`
- `runTasks(): Mono<Long>`
```
The AsyncTaskBarrier Class API

- The AsyncTaskBarrier API contains methods that register, unregister, & (a)synchronously run tasks
  - It provides methods that register & unregister tasks passed as Suppliers
  - It also provides a method that runs all registered tasks (a)synchronously
    - This method doesn’t block
    - When combined with Mono.block() the calling thread won’t exit until all asynchronous task processing completes

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#block](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#block)
Applying the AsyncTask Barrier in Practice
Applying the AsyncTaskBarrier in Practice

```java
//@
public void testExceptions() {
    // Create local variables so that unregister() works properly.
    Supplier< Mono<Void>> syncThrowExceptions = this::syncThrowException;
    Supplier< Mono<Void>> syncNoThrow = this::syncNoThrow;
    Supplier< Mono<Void>> onErrorResume1 = this::onErrorResume1;
    Supplier< Mono<Void>> onErrorResume2 = this::onErrorResume2;

    // Register all the local variables that contain method references.
    AsyncTaskBarrier.register(syncThrowExceptions);
    AsyncTaskBarrier.register(syncNoThrow);
    AsyncTaskBarrier.register(onErrorResume1);
    AsyncTaskBarrier.register(onErrorResume2);

    // Directly register a method reference.
    AsyncTaskBarrier.register(this::syncNoThrow);
}
```

See Reactive/flux/ex4/src/test/java/utils/AsyncTaskBarrierTests.java
End of Implementing the AsyncBarrierTask Framework Using Project Reactor (Part 1)